Extending a Bridge pattern implementation

Introduction：

The bridge pattern is a structural design pattern that aims to separate the abstract part from its implementation part so that the two can change independently. This pattern avoids the complexity brought by multi-level inheritance by introducing a bridge interface.

In the Java example provided by the Refactoring Guru website, the bridge pattern is applied to the relationship between devices (such as TVs and radios) and remote controls. The device class (Device) is the implementation part, and the remote control class (Remote) is the abstract part. The two are decoupled through a composition relationship.

The original implementation:

https://refactoring.guru/design-patterns/bridge/java/example

New Functionality：

To extend the original functionality, I introduced a new device type SmartTV and a new remote control type SmartRemote.

SmartTV: Based on traditional TV, it adds the function of browsing the Internet (browseInternet method).

SmartRemote: Based on the Basic Remote Control, it adds the voice control function (voiceControl method).

The motivation for introducing these new functions is to simulate the characteristics of modern smart devices and make the system closer to real-life application scenarios.

Implementation:

The following is the code implementation of the new SmartTV class:

```
public class SmartTV implements Device {
    private boolean on = false;
    private int volume = 30;
    private int channel = 1;

    @Override
    public boolean isEnabled() {
        return on;
    }

    @Override
    public void enable() {
        on = true;
    }

    @Override
    public void disable() {
        on = false;
    }

    @Override
    public int getVolume() {
        return volume;
    }

    @Override
    public void setVolume(int percent) {
        volume = percent;
    }

    @Override
    public int getChannel() {
        return channel;
    }

    @Override
    public void setChannel(int channel) {
        this.channel = channel;
    }

    @Override
    public void printStatus() {
        System.out.println("------------------------------------");
        System.out.println("| I'm Smart TV set.");
        System.out.println("| I'm " + (on ? "enabled" : "disabled"));
        System.out.println("| Current volume is " + volume + "%");
        System.out.println("| Current channel is " + channel);
        System.out.println("------------------------------------\n");
    }

    public void browseInternet() {
        if (on) {
```

```
            System.out.println("Browsing the internet on SmartTV.");
        } else {
            System.out.println("Cannot browse internet. SmartTV is off.");
        }
    }
}
```

The following is the code implementation of the new SmartRemote class:

```
public class SmartRemote extends BasicRemote {
    public SmartRemote(Device device) {
        super(device);
    }

    public void voiceControl(String command) {
        System.out.println("Voice command received: " + command);
        // Implement command parsing and execution logic here
    }
}
```

Verification:

In order to verify the correctness of the new function, I wrote the following test code:

```
public class Main {
    public static void main(String[] args) {
        testDevice(new Tv());
        testDevice(new Radio());

        System.out.println("Testing Smart Device with Smart Remote:");
        testSmartDevice();
    }
    public static void testSmartDevice() {
        Device smartTV = new SmartTV();
        SmartRemote smartRemote = new SmartRemote(smartTV);

        smartRemote.power();
        smartRemote.volumeUp();
        smartRemote.voiceControl("Open YouTube");
        ((SmartTV) smartTV).browseInternet();
        smartTV.printStatus();
    }

    public static void testDevice(Device device) {
        System.out.println("Tests with basic remote.");
        BasicRemote basicRemote = new BasicRemote(device);
        basicRemote.power();
        device.printStatus();

        System.out.println("Tests with advanced remote.");
        AdvancedRemote advancedRemote = new AdvancedRemote(device);
```

```
        advancedRemote.power();
        advancedRemote.mute();
        device.printStatus();
    }
}
```

The running result should display the status information of SmartTV and simulate the functions of voice control and browsing the Internet.

```
Testing Smart Device with Smart Remote:
Remote: power toggle
Remote: volume up
Voice command received: Open YouTube
Browsing the internet on SmartTV.
-----------------------------------
| I'm Smart TV set.
| I'm enabled
| Current volume is 40%
| Current channel is 1
-----------------------------------
```

Conclusion:

By introducing SmartTV and SmartRemote, I have successfully extended the original bridge mode implementation and added the features of modern smart devices. This design keeps the abstraction and implementation separate, so that the system does not need to modify the existing code when adding new functions, which improves the scalability and maintainability of the system.

During the design process, I considered the realism of the functions and the scalability of the system, and chose to add new functions through inheritance and combination instead of modifying the structure of the original class.